Theoretical Computer Science 90 (1991) 151–170
Elsevier

An algebra of concurrent non-deterministic processes

Ludmila A. Cherkasova and Vadim E. Kotov

Institute of Informatics Systems, Siberian Division of the USSR Academy of Science, 630090, Novosibirsk, USSR

Abstract

Cherkasova, L.A., and V.E. Kotov, An algebra of concurrent nondeterminstic processes, Theoretical Computer Science 90 (1991) 151–170.

This paper illustrates how early ideas and simple naive concepts of concurrency theory of the 1960s have now turned into complex and subtle problems of modern calculi of concurrent processes. An algebra of finite processes AFP_1 is discussed as an example of research which incorporates various aspects of typical process calculi: axiomatization of the proposed algebra, adequate description of "true concurrency" semantics for processes, compositionality and full abstractness in the algebra, handling exceptional situations, etc.

Introduction

In 1963, the second author of this paper and his colleagues came to Andrei Petrovich Ershov, at that time the Head of the Programming Laboratory of the Computing Center in Akademgorodok, asking whether he would be interested in hiring ambitious young people who wanted to start research in the theory of computational parallelism and its application to programming. At that time, the general presentiment of a new technological leap, caused by the fast progress in microelectronics, raised the first wave of discussions and conceptual papers about future parallel computers and parallel programming. These first concepts were mostly based on straightforward, "naive" ideas and on verisimilar reasoning about concurrency which, as we now know, has turned out to be a very exciting and complex area of research. We explained to A.P. Ershov that we want to develop a General Theory of Parallel Systems and Processes and to start with a quite general model of computation, namely with Asynchronous Parallel Processes over Shared Memory. (In this model [11], a parallel program was just a set of statements preceded by guards which we called trigger-functions at that time. Any statement can be initiated any time when its guard's value is proved to be true and terminates any time it wants to terminate. Applying different disciplines of guard checking, different particular models of parallel computation can be formed.)

It looked like A.P. (as we used to call him) was a little bit skeptical about the whole idea at the very beginning. But he was a person very easily carried away by new concepts and had a gift of finding proper ways of implementing some very general and vague ideas into concrete deeds. He proposed particular problems to be solved: to elaborate a formal notion of the (asynchronous) parallel program scheme and to study transformation of sequential program schemes into parallel ones (he proposed to refer to this transformation as *desequention*).

That research has shown that automatic desequention is a doubtful perspective because of the complexity of the analysis of control and data dependencies between statements and procedures in programs. As a result we started to think about an experimental parallel programming language that would be useful for the specification of various forms of "natural" parallelism incorporated in users' tasks and could be efficiently implemented for parallel computers of different architectures. Such a language (we called it BPL, *Basic Parallel Language* [9]) should have a special *control sublanguage* allowing us to describe the diversity of control structures in parallel programs. The control sublanguage contains control operations, functions and expressions. To have the possibility of describing the semantics of the control sublanguage, *Structured Net Algebra* (SNA) based on *Petri nets* was introduced [8].

(A Petri net is a net (graph) with two disjoint sets of vertices: *transitions* which model events in concurrent systems and *places* which serve to model distributed states of the systems. An arc in a net can connect only a place with a transition (the place is an *input place* of the transition) or a transition with a place (called its *output place*). A transition can *fire* (modelling some local activity of a distributed system) if each of its input place has at least one token. When firing, the transition takes a token from each input place and adds a token to each output place. For formal definitions of nets and related notions see, for example, [6, 15, 17].)

The expressions of SNA defining "well-structured" Petri nets are composed using *net operations* such as *net concatenation, exclusion, parallel composition, net iteration* and *marking* [8].

SNA is an algebra for modelling primarily structural properties of abstract distributed concurrent systems and it has proved to be a convenient specification tool because nets, underlying SNA, have reasonably good descriptive abilities (particularly, the nets extended by mechanisms to describe hierarchy and refinement [3]). However, although providing a good insight into the structural properties of concurrent systems, SNA does not contain sufficient support for the derivation of their behavioural properties. Hence, to study subtle interrelations between structural and behavioural properties of asynchronous systems, we have become involved in the study of algebraic modelling of concurrent processes which has become an increasingly vigorous research area during the last decade. The most popular and fully developed theories are now *Net Theory* originating in the work of C.A. Petri, *Calculus of Communicating Systems* (CCS) proposed by R. Milner [12] and *Theory of Communicating Sequential Processes* (TCSP) initiated by C.A.R. Hoare [1]. Each of these theories describes a *process* as a set of *actions* with the introduction of

relations such as *sequentiality*, *concurrency* or *nondeterminism*. For the cases of "pure sequential" nondeterministic processes (the relation of concurrency is absent) or "pure concurrent" processes (which do not involve the alternative or nondeterminism between actions), there are at least two well known models, such as finite automata and partially ordered sets, that are widely used for representation of these particular types of processes. However, models for processes based on all three relations are far from clear. In particular, the problem of adequate expressing of deep relationships between concurrency and nondeterminism is still open.

We briefly recall the different approaches to expressing concurrency and nondeterminism in different models. Depending on the representation of concurrency, the models can be divided into the following three groups.

(1) The concurrent execution of two processes is simulated by nondeterministic *interleaving of their atomic actions* (i.e. concurrency is simulated by sequential nondeterminism). Therefore, the concurrent execution of two atomic actions a and b could be defined by the following axiom:

$$a \parallel b = ab \text{ or } ba,$$

where $\parallel$ denotes concurrent execution of a and b, ab means that a precedes b, ba means that b precedes a. However, if we consider a system with two mutually exclusive actions a and b (let us denote this situation by $a \; mutex \; b$), then the behaviour of such a system can be specified by the same axiom:

$$a \; mutex \; b = ab \text{ or } ba.$$

Thus, the processes $(a \parallel b)$ and $(a \; mutex \; b)$ are indistinguishable in the models based on interleaving semantics. If the behaviour of concurrent systems and processes is characterized by sequences of totally ordered actions, then the concurrency operator (or relation) is not primitive:

$$concurrency = sequentiality + nondeterminism.$$

(2) The concurrent execution of two processes is simulated by *interleaving of multisets of their atomic actions*. For example, the concurrent execution of two atomic actions a and b could be defined by the following axiom:

$$a \parallel b = ab \text{ or } ba \text{ or } a|b,$$

i.e. a precedes b, or b precedes a, or a and b occur at the same time (the last situation is denoted as *co-occurrence* $a|b$). Hence, the concurrency operator (or relation) is also not primitive in these models:

$$concurrency = sequentiality + nondeterminism + co\text{-}occurrence.$$

(3) The third group of concurrent models is based on the concept of the process behaviour as a *partially ordered set of actions*. Such models as *Pomsets* (partially ordered multisets) [16], *Causal (Occurrence) Nets* [13, 14], *Event Structures* [13], *A-nets* [10] are intended to describe *true concurrency*. The precedence relation on

actions is defined as the *causal dependence* of actions in these models. This relation induces some partial order on actions. Accordingly, two actions are concurrent if they are casually independent. Thus, a concurrent process, the elements of which are partially ordered by the precedence relation, can be explicitly represented by a partially ordered set (poset) (see [7, 14, 16]). The behaviour of a concurrent nondeterministic system is described by a set of its "pure concurrent" processes. Each process in such a set is a result of a nondeterministic choice among conflicting actions during a run of the system. However, often it is necessary and preferable to deal with conflicts on a semantical level and to express the behaviour of a system with conflicts as some unique integral semantic object. For this reason, Event Structures, Occurrence Nets and A-nets generalize processes by augmenting posets with the conflict [13] or alternative relation [10]. Two actions a and b are alternative if the occurrence of a excludes the occurrence of b, and vice versa.

The known models of processes can be parted into different groups depending also on the way they represent *nondeterminism.* For example, the semantic models for CCS and TCSP have a common feature in the representation of concurrency as interleaving of actions, but the semantic representation of nondeterminism is different in these calculi.

(1) Nondeterminism is a basic relation in *action trees* for CCS. The arcs (labelled by action symbols) issued from a node offer to perform their actions, as alternatives and, in such a way, nondeterminism is explicitly represented on the semantic level of CCS.

(2) In TCSP, every process is characterized by a so-called *refusal set.* Each refusal set consists of a set of *failures.* A failure is a pair (σ, V), where σ is a finite sequence of actions in which the process may have been engaged up to a certain moment and V is a set of actions which the process is able to reject on the next step. In other words, a process is defined as a set of possible execution sequences each of which is augmented by some "negative" information.

Thus, nondeterminism is not a basic relation on the semantic level of TCSP. However, the "negative" part added to each execution sequence of the process on the semantic level gives necessary information about nondeterminism specified by initial algebraic process formula. This model can be considered as an interesting and remarkable example, illustrating how nondeterminism (alternative) can be represented and investigated without introducing it explicitly on the semantic level.

If we introduce two axes for representing concurrency and nondeterminism, then different models of processes can be placed as shown in Fig. 1 according to increasing expressiveness of these relations.

The rest of the paper discusses the algebra AFP_1 with a semantical model based on *posets with non-actions.*

As one can see in Fig. 1, there is no shortage of denotational models for concurrent nondeterministic processes. In the face of such an abundance, it is better to present some motivation for introducing new such models. It should be noted that the notion of a process is used in two senses: (i) as a *specification* of a "dynamic" object

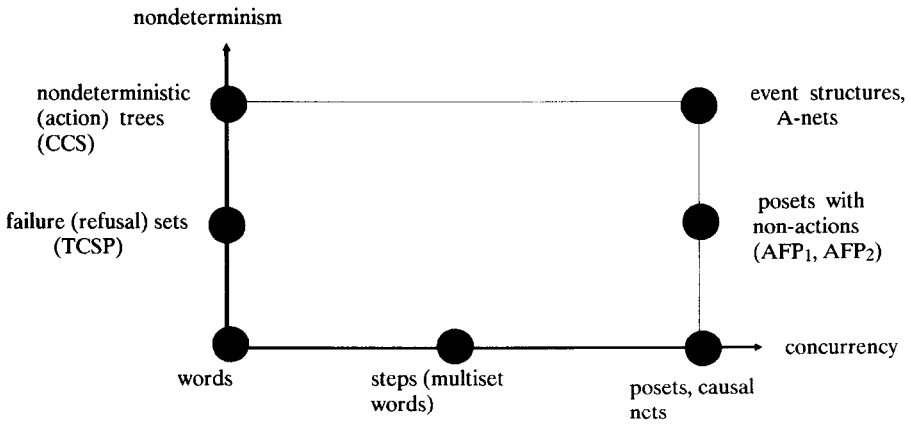

Fig. 1.

by means of some formalism, and (ii) as a *behaviour* (or semantics) of a specified dynamic object. These two meanings are different in some theories. Their identity is highly desirable in the elaboration of practical tools for the verification and syntheses of systems.

Algebras proposed mostly for process specification could be referred to as "*descriptive*" algebras. In descriptive algebras, a process specification provides a good insight into the structural properties of designed concurrent systems. "*Analytical*" algebras contain sufficient support for the validation of the behavioural properties. It is desirable to elaborate a process algebra which is both descriptive and analytical.

The first and the main goal of introducing AFP_1 is to bridge the gap between the descriptive and analytical theories of concurrent processes.

The second and more partial goal consists of the following. It is striking that most of the interleaving models are supported by a variety of algebraic results in the form of, for example, full abstractness with respect to some notion of operational behaviour or equational proof systems; this is in contrast to noninterleaving models for which you will find very few such algebraic results. In this paper, we would like to present some full abstractness and completeness results for AFP_1.

1. Algebra AFP_1 of finite (nondeterministic concurrent) processes

Let $\mathcal{A} = \{a, b, c, \ldots\}$ be a finite alphabet of action symbols (the *action basis* of a process).

Elementary actions can be combined into a composite process in AFP_1 using the operations of *precedence* ";", *exclusive or* (or *alternative*) "∇" and *concurrency* "$\|$". Intuitively, the process $(a; b)$ performs, at first, the action a and only after that it performs the action b. The process $(a \nabla b)$ consists of two possible behaviours: if it chooses the action a then the action b does not occur, and vice versa. The formula $(a \| b)$ specifies the process in which the actions a and b occur concurrently.

We suppose that each action has its own unique name. Thus, if we have a process P consisting of different subprocesses P_1 and P_2 such that an action symbol c occurs in both P_1 and P_2, then the performance c in P should be synchronized by the performances of c in P_1 and in P_2 simultaneously, i.e. the process P can perform the action c if and only if both subprocesses P_1 and P_2 are ready to perform the action c. For example, the process formula $(a;c)\|(b;c)$ specifies the process in which the actions a and b are performed concurrently and only after that (i.e. after both actions a and b have been executed) can the action c be performed. Thus, if some action in one process needs an action in another process for an actual synchronized execution (it is a typical situation for communicating processes) then this situation can be easily specified in AFP_1 just by using the same symbol for both actions. Such an approach allows us not to restrict the number of communicating processes (unlike, for example, CCS).

Since we intend to construct the algebra which combines the mechanism to specify both processes and their properties, the process specification (process formula) is twofold: on the one hand, it specifies possible process behaviours; on the other hand, it can be considered as defining a set of properties the process enjoys.

The semantics of a process described by a formula of AFP_1 is defined as a set of partial orders (see next section). However, as has been mentioned earlier, such a semantic representation of concurrent nondeterministic processes involves only two basic relations between process elements: precedence and concurrency. The information about an alternative relation between elements of the initial processes is lost. To represent (implicitly) the alternative relation on the semantic level, we introduce some "negative" information about those potential process actions which have not been chosen to be performed during the process functioning (the idea, in some sense, is similar to failure semantics for TCSP). Thus, we introduce the dual to the $\mathcal{A}$ alphabet of the "negated" symbols $\bar{\mathcal{A}} = \{\bar{a}, \bar{b}, \bar{c}, \dots\}$ for denoting "*non-actions*", i.e. the symbols which point to the fact that the correspondent actions do not occur in this run, because they were not chosen among alternative actions.

We are going to define the alternative operation ∇ in a "very structural" way. The semantics of a process described by an AFP_1 formula will be defined as a set of partial orders. Thus, a process described by the formula $(a \nabla b)$ will be characterized by two partial orders: the first one defines the process behaviour if the action a is chosen to be performed and the second one defines the process behaviour if the action b is chosen.

We would like to have more complete information about nondeterminism in a process structure at the semantic level of the partial order representation. We add some "negative" information to our consideration and reasoning about processes. In particular, we would like to know which actions have not been chosen during the concrete process behaviour. To denote the fact that an action a does not occur in some process run (because some alternative action has been performed), we introduce the negated symbol $\bar{a}$ and call it a *non-action*. So, the process $(a \nabla b)$ is characterized by the following behaviours: in the first behaviour, the action a occurs

and the non-action $\bar{b}$ appears; in the second one, the action b occurs and, additionally, the non-action $\bar{a}$ appears.

Thus, each partial order representing one of the possible process behaviours has an "observable" (positive) part and an "unobservable" (negative) one. The "positive" part consists of the process actions whch have been performed during this process run. The "negative" part consists of the non-actions which have not been executed (have not been chosen) during this process functioning.

However, there exists another reason why some actions could not be performed during some process functioning. Let us consider a process defined by the following formula

$$A = (a \parallel b) \parallel (a \nabla b).$$

This process specification consists of two subformulas $B = (a \parallel b)$ and $C = (a \nabla b)$. The subformula $B = (a \parallel b)$ specifies a process in which both actions a and b should be performed and performed concurrently. The subformula $C = (a \nabla b)$ defines two possible methods of process functioning:

(1) a occurs and b does not occur (i.e., the non-action $\bar{b}$ takes place), or

(2) b occurs and a does not occur (i.e., non-action $\bar{a}$ takes place).

If we try to define a process specified by the formula A as a common behaviour of processes specified by B and C, then we discover that there exists no common possible behaviour because each combination of requirements of B and C is *contradictory.* It is required in B that both actions a and b should occur but, on the other hand, C requires that b cannot occur if the alternative action a is chosen to be performed. A similar situation occurs concerning the action a. In such situations we will say that the action b (or, correspondingly, the action a) is *deadlocked.* To denote the deadlocked actions, we introduce the special alphabet $\Delta_{\mathscr{A}} = \{\delta_a, \delta_b, \delta_c, \ldots\}$.

In addition to the operations ";" (precedence), "∇" (alternative) and "$\parallel$" (concurrency), we introduce three more operations: "$\vee$" (*disjunction* or *union*), "$\neg$" (*not occur*) *and* "$\tilde{\neg}$" (*mistaken not occur*).

Intuitively, the formula $(A \vee B)$ defines a process in which either the subprocess defined by A or the subprocess defined by B occurs, i.e. the set of possible process behaviours defined by $(A \vee B)$ is the union of the sets of process behaviours defined by A and B. The operation $\neg$ is a modified negation: $\neg A$ means that the process A does not occur, i.e. no action of A is executed. The operation $\tilde{\neg}$ is another type of negation: $\tilde{\neg} A$ means that the process A does not occur as a result of some mistake, i.e. any action of A does not occur in a process functioning as a result of some contradictory requirements in a process specification.

So, a formula of AFP_1 in a basis $\mathscr{A} \cup \bar{\mathscr{A}} \cup \Delta_{\mathscr{A}}$ is a term of the following language:

$$A ::= a \mid \bar{a} \mid \delta_a \mid (A \| A) \mid (A \nabla A) \mid (A; A) \mid (A \vee A) \mid \neg A \mid \tilde{\neg} A$$

where $a \in \mathscr{A}$, $\bar{a} \in \bar{\mathscr{A}}$ and $\delta_a \in \Delta_{\mathscr{A}}$.

2. Denotational semantics

The semantics of AFP_1 formulae will be characterized by the sets of partial orders in the alphabet $\mathscr{A} \cup \bar{\mathscr{A}} \cup \Delta_{\mathscr{A}}$.

A *partially ordered set* (*poset*) is a pair $p = (V, <)$ consisting of

(i) a vertex set V typically modelling process actions, i.e. $V \subseteq \mathscr{A} \cup \bar{\mathscr{A}} \cup \Delta_{\mathscr{A}}$.

(ii) a partial order relation $<$ over V, with $a < b$ typically interpreted as the action a necessarily preceding the action b.

Let us denote the *action subset* of V by $V^+ = \{x \in V \mid x \in \mathscr{A}\}$ and the *non-action subset* of V by $V^- = \{x \in V \mid x \in \bar{\mathscr{A}}\}$.

In this paper we will consider the posets which satisfy the following conditions:

(1) a and $\bar{a}$ do not occur in a poset p together, i.e. either the action a occurs in p and occurs exactly once or the non-action $\bar{a}$ occurs in p once;

(2) if there exists some deadlocked action δ_a such that $\delta_a \in V$ then $V \subseteq \Delta_{\mathscr{A}}$.

(3) the partial order relation $<$ over V is irreflexive;

(4) $\forall \bar{a} \in V^- \neg \exists x \in V: (x < \bar{a}) \vee (\bar{a} < x)$, i.e. all non-actions are incomparable by $<$.

Now, we will comment on these conditions. As has been noted earlier, each action in a process formula has its own unique name. Three different situations (excluding each other) can arise during a process functioning:

(1) either the action a occurs (exactly once) or the action a does not occur and we distinguish two different reasons for such a situation arising:

(2) the non-action $\bar{a}$ takes place when some action alternative to a occurs, or

(3) the deadlocked action δ_a takes place when the action a cannot occur as a result of some mistake (contradiction) in a process specification.

The example of a contradictory specification $(a \| b) \| (a \nabla b)$ we have considered earlier. In other words, specifications like $(a \| \bar{a})$ lead to the situation where the action a is deadlocked (i.e., δ_a takes place). If we have the requirement $(a; a)$ in a process specification then the uniqueness of each action name leads to a similar contradiction (i.e. as a result the deadlocked action δ_a takes place). That is the reason why we demand that partial order relations over V be irreflexive. So, if some semantical contradiction is discovered in a partial order representing one of the possible process behaviours, then we announce this partial order (i.e. corresponding process behaviour) as a contradictory one, denote all of its actions as deadlocked ones and eliminate it from our further consideration.

Thus, a poset $p = (V, <)$ is

(1) either *degenerate*: $p = (V, \emptyset)$, where $V \subseteq \Delta_{\mathscr{A}}$,

(2) or it consists of two parts: the "positive" part containing actions of V^+, and the "negative" one containing non-actions of V^-. Moreover, $(< \cap (V^+ \times V^+)) = <$ and $(< \cap (V^- \times V^-)) = \emptyset$.

To define the denotational semantics of the basic process operations we will introduce the similar poset operations: ";" (precedence), "∇" (alternative), "$\|$" (concurrency), "$\neg$" (*not occur*), "$\tilde{\neg}$" (mistaken not occur).

If the poset p constructed by means of these operations does not satisfy the conditions (1)-(4) mentioned above, we "correct" it using the auxiliary *regularization operation* $[p]$:

$$[p]=\begin{cases} p & \text{if } p \text{ is a poset satisfying the conditions (1)-(4),} \\ (\Delta_{\mathscr{A}(V)},\emptyset) & \text{otherwise,} \end{cases}$$

where $\mathscr{A}(V)=\{a\,|\,(a\in V)\vee(\bar{a}\in V)\vee(\delta_a\in V)\}$, i.e. if some semantical contradiction arises in the resulting poset p constructed by means of poset operations, then all the actions of p are deadlocked and cannot occur.

The *precedence* of two posets $p_1=(V_1,<_1)$ and $p_2=(V_2,<_2)$ is defined as follows:

$$p=(V_1,<_1);(V_2,<_2)=[(V_1\cup V_2,<_1\cup<_2\cup(V_1^+\times V_2^+))],$$

i.e. in the new poset p every action of p_1 precedes every action of p_2 or, if the constructed object does not satisfy the conditions (1)-(4), then p is the degenerate poset.

Example. Let $p_1=(\{a\},\emptyset)$ and $p_2=(\{b\},\emptyset)$. Then $p_3=(p_1;p_2)=(\{a,b\},<_3)$, where $a<_3 b$. However, $p_4=(p_3;p_1)=(\{\delta_a,\delta_b\},\emptyset)$.

The *concurrency* $\|$ of two posets $p_1=(V_1,<_1)$ and $p_2=(V_2,<_2)$ is defined as follows:

$$p=(V_1,<_1)\,\|\,(V_2,<_2)=[(V_1\cup V_2,(<_1\cup<_2)^*)],$$

where $(<_1\cup<_2)^*$ is the transitive closure of the relation $(<_1\cup<_2)$.

Example. Let $p_1=(\{a,c\},<_1)$, where $\alpha<_1 c$ and $p_2=(\{b,c\},<_2)$, where $b<_2 c$. Then $p_3=(p_1\,\|\,p_2)=\{(\{a,b,c\},<_3)$, where $a<_3 c$, $b<_3 c$. Let us consider additionally: $p_4=(\{a,c\},<_4)$, where $c<_4 a$. Then $(p_3\,\|\,p_4)=(\{\delta_a,\delta_b,\delta_c\},\emptyset)$.

To define the next operation we introduce the following notion of the *modified union* $\tilde{\cup}$:

$$(\{p_1\}\,\tilde{\cup}\,\{p_2\})=\begin{cases} \{p_1\} & \text{if } p_2 \text{ is a degenerate poset,} \\ \{p_2\} & \text{if } p_1 \text{ is a degenerate poset,} \\ \{\{p_1\},\{p_2\}\} & \text{otherwise.} \end{cases}$$

The modified union for posets absorbs the degenerate posets, i.e. the posets all the actions of which are dealocked. In such a way, we eliminate the posets constructed by contradictory specifications.

The *alternative* ∇ of two posets $p_1=(V_1,<_1)$ and $p_2=(V_2,<_2)$ is defined as follows:

$$\begin{aligned} p&=(V_1,<_1)\,\nabla\,(V_2,<_2)\\ &=\{[(V_1\cup\bar{V}_2,<_1)]\}\,\tilde{\cup}\,\{[(\bar{V}_1\cup V_2,<_2)]\}, \end{aligned}$$

where $\bar{V}=\{\bar{a}\,|\,(a\in V)\vee(\bar{a}\in V)\vee(\delta_a\in V)\}$.

It should be noted that $(p_1\nabla p_2)$ is not a poset, but a set of two posets describing possible alternative computations (alternative process runs).

Example. Let $p_1 = (\{a\}, \emptyset)$ and $p_2 = (\{b\}, \emptyset)$. Then

$$(p_1 \nabla p_2) = \{(\{a, \bar{b}\}, \emptyset) \tilde{\cup} \{(\{\bar{a}, b\}, \emptyset)\}.$$

The modified negations are defined as follows.

Operation $\neg$: $\neg p = \neg(V, <) = (\bar{V}, \emptyset)$, i.e. no action of p occurs because some alternative actions occur.

Operation $\tilde{\neg}$: $\tilde{\neg} p = \tilde{\neg}(V, <) = (\Delta_{\mathscr{A}(V)}, \emptyset)$, i.e. all the actions of p are deadlocked and cannot happen as a result of some mistake in a process specification.

We extend the operations intorduced above for sets of partial orders in the natural way. Let $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_k\}$ be sets of partial orders, then

$$P \otimes Q = \bigcup_{j=1}^{k} \left(\bigcup_{i=1}^{n} \{p_i \otimes q_j\} \right), \text{ where } \otimes \in \{;, \|, \nabla\}.$$

Similarly, $\neg P = \bigcup_{i=1}^{n} (\neg p_i)$ and $\tilde{\neg} P = \bigcup_{j=1}^{n} (\tilde{\neg} p_i)$.

The denotational semantics of the AFP_1 formulae is defined as follows:

(1) $\mathbf{D}[a] = (\{a\}, \emptyset)$, $\mathbf{D}[\bar{a}] = (\{\bar{a}\}, \emptyset)$, $\mathbf{D}[\delta_a] = (\{\delta_a\}, \emptyset)$,
(2) $\mathbf{D}[(A \| B)] = (\mathbf{D}[A] \| \mathbf{D}[B])$,
(3) $\mathbf{D}[(A; B)] = (\mathbf{D}[A]; \mathbf{D}[B])$,
(4) $\mathbf{D}[(A \nabla B)] = (\mathbf{D}[A] \nabla \mathbf{D}[B])$,
(5) $\mathbf{D}[(A \vee B)] = (\mathbf{D}[A] \tilde{\cup} \mathbf{D}[B])$,
(6) $\mathbf{D}[\neg A] = \neg \mathbf{D}[A]$,
(7) $\mathbf{D}[\tilde{\neg} A] = \tilde{\neg} \mathbf{D}[A]$.

3. Full abstractness

One of the natural ways of reasoning is to identify two processes if they generate the same sets of possible behaviours. Each behaviour in such a set is a result of some nondeterministic choice among alternative actions during a run of the process. Thus, each process behaviour can be considered as a poset consisting of the process actions. If $p = (V, <)$ is a poset and $V \subseteq \mathscr{A} \cup \bar{\mathscr{A}} \cup \Delta_{\mathscr{A}}$ then let $p^+ = (V^+, <)$ denote its positive, "observable" part over the action subset. Correspondingly, if a process defined by A is characterized by the set of partial orders $\mathbf{D}[A] = \{p_1, \ldots, p_n\}$ then let $\mathbf{D}^+[A] = \{p_1^+, \ldots, p_n^+\}$ denote its positive part over the action subset.

Two processes A and B are *observationally equivalent* ($A \approx_+ B$) iff $\mathbf{D}^+[A] = \mathbf{D}^+[B]$.

A context $\mathbf{C}[\ ]$ is an expression with zero or more "holes" to be filled by an expression. We write $\mathbf{C}[A]$ for the result of placing A in each "hole",

Denotational semantics (or model) $\mathbf{D}$ is called *fully abstract with respect to equivalence relation* $\approx_+$ iff

$$A, B \in \mathrm{AFP}_1: \quad \mathbf{D}^+[A] = \mathbf{D}^+[B] \Leftrightarrow \forall \mathbf{C}[\ ]: \quad \mathbf{C}[A] \approx_+ \mathbf{C}[B].$$

Theorem 1. *Denotational semantics* **D** *is fully abstract with respect to* $\approx_+$.

By the definition of $\approx_+$, the following two processes $A=(a\nabla b)$ and $B=(a\vee b)$ are observationally equivalent because $\mathbf{D}^+[A]=\mathbf{D}^+[B]$ and cannot be distinguished by any observation of partially ordered actions. However, if we consider a process defined by the formula $((a\nabla b)\|a)$ and substitute $A=(a\nabla b)$ for $B=(a\vee b)$ then we obtain behaviourally different processes:

$$\begin{aligned}\mathbf{D}[(a\nabla b)\|a]&=\mathbf{D}[(a\nabla b]\|\mathbf{D}[a]\\&=[(\{a,\bar{b})\},\emptyset)]\,\breve{\cup}\,[(\{a,b,a\},\emptyset)]\\&=(\{a,b)\},\emptyset)\,\breve{\cup}\,(\{\delta_a,\delta_b\},\emptyset)=(\{a,\bar{b})\},\emptyset),\\\mathbf{D}[(a\vee b)\|a]&=\mathbf{D}[(a\vee b)]\|\mathbf{D}[(a]=\{(\{a\},\emptyset),(\{a,b\},\emptyset)\},\end{aligned}$$

and

$$\mathbf{D}^+[(a\nabla b)\|a]\neq\mathbf{D}^+[(a\vee b)\|a].$$

Thus, observationally equivalent processes can become nonequivalent if they are placed into the same process context (or the same process environment), i.e. $\approx_+$ is not a congruence.

So, if we would like to consider the equivalent processes as modules that can be mutually exchanged in any context without affecting the observable behaviour of the latter, we need both "positive", "observable" information about the actions which a process can perform (during its possible behaviour) and "negative", "invisible" information concerning the actions which cannot occur in this process behaviour.

From this point of view, the operations ∇ and $\vee$ (correspondingly, processes $A=(a\nabla b)$ and $B=(a\vee b)$) are different. Using the operation ∇ (alternative) provided with some "negative" information, we define a process in which a choice of further possible behaviour depends on its environment, The operation $\vee$ (disjunction) defines a process with nondeterministic choice (i.e. with choice independent of the environment).

The fact that $\mathbf{D}[(a\nabla b)]=\{(\{a,\bar{b})\},\emptyset),(\{b,\bar{a})\},\emptyset)\}$ and $\mathbf{D}[(a\nabla b)\|a]=(\{a,\bar{b})\},\emptyset)$ illustrates that a choice among behaviours specified by means of the alternative operation ∇ depends on the environment.

On the other hand, we have $\mathbf{D}[(a\vee b)]=\{(\{a\},\emptyset),(\{b)\},\emptyset)\}$ and $\mathbf{D}[(a\vee b)\|a]=\{(\{a\},\emptyset),(\{a,b\},\emptyset)\}$. Thus, the disjunction $\vee$ corresponds to the nondeterministic choice.

4. Axiomatization for $\mathbf{AFP}_1$

The proposed semantics for the processes of AFP_1 immediately suggests a natural equivalence between them. Two processes specified by formula A and B are *equivalent* ($A\approx_e B$) iff $\mathbf{D}[A]=\mathbf{D}[B]$.

Two processes specified by formulae A and B are *congruent* ($A \approx_{co} B$) iff $\mathbf{C}[A] \approx_e \mathbf{C}[\beta]$ for any context $\mathbf{C}[\ ]$.

Proposition. *If* $A \approx_e B$ *then* $A \approx_{co} B$.

This proposition immediately follows from the definition of semantics of the AFP_1 formulae.

Now, we will propose the axiom system which corresponds to the congruence relation defined above. The following axioms characterize the properties of the operators introduced. Here A, B, C denote formulae of AFP_1, $a \in \mathscr{A}$, $\bar{a} \in \bar{\mathscr{A}}$ and $\delta_a \in \Delta_{\mathscr{A}}$.

(1) *Associativity.*

(1.1) $A \| (B \| C) = (A \| B) \| C$,

(1.2) $A \nabla (B \nabla C) = (A \nabla B) \nabla C$,

(1.3) $A \vee (B \vee C) = (A \vee B) \vee C$,

(1.4) $A ; (B ; C) = (A ; B) ; C$.

(2) *Commutativity.*

(2.1) $A \| B = B \| A$,

(2.2) $A \nabla B = B \nabla A$,

(2.3) $A \vee B = B \vee A$.

(3) *Distributivity.*

(3.1) $(A \| B) ; C = (A ; C) \| (B ; C)$,

(3.2) $A ; (B \| C) = (A ; B) \| (A ; C)$,

(3.3) $(A \vee B) ; C = (A ; C) \vee (B ; C)$,

(3.4) $A ; (B \vee C) = (A ; B) \vee (A ; C)$,

(3.5) $(A \vee B) \| C = (A \| C) \vee (B \| C)$,

(3.6) $A \nabla (B \| C) = (A \nabla B) \| (A \nabla C)$.

(4) *Axioms for* ∇ *and* $\neg$.

(4.1) $A \nabla B = (A \| \neg B) \vee (\neg A \| B)$.

(4.2) $\neg (A \| B) = \neg A \| \neg B$,

(4.3) $\neg (A \vee B) = \neg A \vee \neg B$,

(4.4) $\neg (A ; B) = \neg A \| \neg B$,

(4,5) $\neg a = \bar{a}$,

(4.6) $\neg \bar{a} = \bar{a}$,

(4.7) $\neg \delta_a = \bar{a}$.

(5) Structural properties.

(5.1) $\bar{a} ; A = \bar{a} \| A$,

(5.2) $A ; \bar{a} = \bar{a} \| A$,

(5.3) $A \| (A ; B) = A ; B$,

(5.4) $B \| (A ; B) = A ; B$,

(5.5) $A ; B ; C = (A ; B) \| (B ; C)$,

(5.6) $(A ; B) \| (B ; C) = (A ; B) \| (B ; C) \| (A ; C)$,

(5.7) $A \| A = A$,

(5.8) $A \vee A = A$.

(6) Axioms for deadlocked actions and $\tilde{\neg}$.

(6.1) $a \| \bar{a} = \delta_a$,

(6.2) $a ; a = \delta_a$,

(6.3) $\delta_a ; A = \delta_a \| \tilde{\neg} A$,

(6.4) $A ; \delta_a = \delta_a \| \tilde{\neg} A$,

(6.5) $A \| \delta_a = \delta_a \| \tilde{\neg} A$,

(6.6) $\tilde{\neg} a = \delta_a$,

(6.7) $\tilde{\neg} a = \delta_a$,

(6.8) $\tilde{\neg} \delta_a = \delta_a$,

(6.9) $\tilde{\neg}(A \| B) = \tilde{\neg} A \| \tilde{\neg} B$,

(6.10) $\tilde{\neg}(A ; B) = \tilde{\neg} A \| \tilde{\neg} B$,

(6.11) $\tilde{\neg}(A \vee B) = \tilde{\neg} A \vee \tilde{\neg} B$,

(6.12) $\tilde{\neg} A \vee \tilde{\neg} B = \tilde{\neg} A$, if $\mathscr{A}(A) \subseteq \mathscr{A}(B)$ (see Section 5),

(6.13) $A \vee \tilde{\neg} B = A$, if A is a normal $\|$-conjunct (see Section 5).

This set of axioms is *sound* for $\approx_{co}$, i.e. $A = B$ is an instance of an axiom from the set above then $A \approx_{co} B$. The proof consists of determining the semantics of A and B, whatever they are, and comparing them. It can be done directly by the definitions of operators. In order to prove that the axiom set completely characterizes the congruence we have to introduce a canonical form for the AFP_1 formulae.

5. Canonical form of AFP_1 formulae

Let us denote by $\mathscr{A}(A)$ the subset of symbols of $\mathscr{A}$ occurring in a formula A. More exactly:

$$\mathscr{A}(a) = a, \qquad \mathscr{A}(\bar{a}) = a, \qquad \mathscr{A}(\delta_a) = a,$$

$$\mathscr{A}(A \otimes B) = \mathscr{A}(A) \cup \mathscr{A}(B), \quad \text{where } \otimes \in \{;, \|, \nabla, \vee\},$$

$$\mathscr{A}(\neg A) = \mathscr{A}(A), \qquad \mathscr{A}(\tilde{\neg} A) = \mathscr{A}(A).$$

Also let $\bar{\mathscr{A}}(A) = \{\bar{a} \mid a \in \mathscr{A}(A)\}$, $\Delta(A) = \{\delta_a \mid a \in \mathscr{A}(A)\}$ and $\hat{\mathscr{A}}(A) = \mathscr{A}(A) \cup \bar{\mathscr{A}}(A) \cup \Delta(A)$.

A formula over the joined alphabet $\mathscr{A} \cup \bar{\mathscr{A}}$ which contains only the operators of concurrency "$\|$" and precedence ";" is called a *$\|$-conjunctive term.* A $\|$-conjunctive term is called a *normal $\|$-conjunct*, if it has the form $(A_1 \| A_2 \| \cdots \| A_n)$ and the following requirements are valid:

(1) every formula A_i $(1 \leqslant i \leqslant n)$ is an elementary formula $a \in \mathscr{A}$, or $\bar{a} \in \bar{\mathscr{A}}$, or an elementary precedence $(a ; b)$, where $a, b \in \mathscr{A}$ and $a \neq b$;

(2) for any formulae A_i and A_j $(1 \leqslant i \neq j \leqslant n)$ such that $\mathscr{A}(A_i) \cap \mathscr{A}(A_j) \neq \emptyset$, A_i and A_j have a form of different elementary precedences;

(3) for each pair of formulae $A_i = (a ; b)$ and $A_j = (b ; c)$, where $(1 \leqslant i \neq j \leqslant n)$, there exists a term $A_k = (a ; c)$ describing the transitive closure of the precedence relation for the actions a, b, c.

A formula A is in the *disjunctive normal* form (or in the *canonical form*) iff either $A = (A_1 \vee A_2 \vee \cdots \vee A_n)$, where A_i $(1 \leqslant i \leqslant n)$ are normal $\|$-conjuncts and all A_i are pairwise different, or $A = \tilde{\neg} B$ (i.e. all the actions of A are deadlocked).

Examples. (1) The formula $(a;b) \| (b;c)$ is a $\|$-conjunctive term but it is not a normal $\|$-conjunct.

Let us verify the requirements (1)-(3) in the definition of the normal $\|$-conjunct. Condition (1) is trivially valid; condition (2) also holds: $\mathcal{A}(a;b) \cap \mathcal{A}(b;c) \neq \emptyset$ and the formulae $(a;b)$ and $(b;c)$ have a form of different elementary precedences. However, condition (3) is not valid: $(a;b) \| (b;c)$ does not contain the term $(a;c)$ describing the transitive closure of the precedence relation for the actions a, b, c.

(2) The formula $(a;b) \| (b;c) \| (a;c)$ is a normal $\|$-conjunct.

(3) The formula $(a \| (a;b))$ is a $\|$-conjunctive term but not a normal $\|$-conjunct because condition (2) in the definition of the normal $\|$-conjunct is not valid.

(4) the formula $(((a \| (a;b;c)) \nabla (d;e)) \| f)$ has the following disjunctive normal form:

$$((a;b) \| (b;c) \| (a;c) \| f \| \bar{d} \| \bar{e}) \vee ((d;e) \| f \| \bar{a} \| \bar{b} \| \bar{c}).$$

It should be noted that each normal $\|$-conjunct characterizes one of the possible alternative process realizations and has a special form coinciding with the partial order representation.

We shall write $A = B$ to mean that the equation may be proved from the above axioms (1.1)-(6.13) by normal equational reasoning. We will prove that every process can be reduced to an equivalent one which is in the canonical form (by using axioms (1.1)-(6.13)).

Two canonical forms A and B are *isomorphic* iff A could be reduced to B (and vice versa) with the help of the commutativity and associativity axioms for the operators $\|$ and $\vee$. For example, the form $(a \| b \| c) \vee (c \| \bar{a} \| \bar{b})$ is isomorphic to the following one: $(\bar{a} \| \bar{b} \| c) \vee (b \| a \| \bar{c})$.

Theorem 2. *Every formula* $A \in \mathrm{AFP}_1$ *can be proved to be equal to its unique (up to isomorphism) canonical form.*

Theorem 3. *For any process formulae A and B of* AFP_1 *the following statement is valid*:

$$A \approx_{\mathrm{co}} B \Leftrightarrow A = B.$$

Thus, any valid equation between processes may be proved from the axiom set.

6. Deduction of process properties

A formula of AFP_1 specifies both a process and its properties. Two main classes of the process properties can be distinguished: *total* and *partial properties*. The

former properties are valid for any actual behaviour of the process; the latter are valid for a subset of possible behaviour. The second class of properties emerges because of the inclusion of alternative actions in generalized processes. Intuitively, the total properties correspond to the notion of the validity of a model, the partial properties correspond to the notion of satisfiability.

Let us consider the process defined by the formula $((a \nabla b) \| c)$. The subformula $(a \nabla b)$ describes the total property of the process, namely the fact that its actions a and b are always alternative. The property specified by the subformula a (i.e. the fact that the action a occurs) is partial because there exists the process behaviour (namely, $(\{b, c, \bar{a}\}, \emptyset)$) in which the action a does not occur, i.e. $\bar{a}$ appears. Similarly, the property $(a \| c)$ which claims that the actions a and c are concurrent is partial and the property described by the formula c (i.e. that the action c occurs in the process behaviour) is total.

Let us introduce some additional notions and definitions.

We denote by $p \lceil W$ the projection of partial order p on the alphabet $W \subseteq V$. It is defined as follows:

$$p \lceil W = (V \cap W, ((V \cap W) \times (V \cap W)) \cap <).$$

The projection $\lceil$ is extended for sets of partial orders in the natural way:

$$(\{p_1\} \cup \{p_2\}) \lceil W = \{p_1 \lceil W\} \cap \{p_2 \lceil W\}.$$

We will use Δ instead of $\Delta_{\mathscr{A}}$ to specify an alphabet of a process all actions of which are deadlocked when the knowledge about $\mathscr{A}$ is not essential. Similarly, we will use the symbol δ to denote a process all the actions of which are deadlocked.

A property specified by a formula B is *partial* for a process defined by a formula A (denotation: $A \models_{\mathrm{p}} B$) iff

(1) $\mathbf{D}[A] \neq (\Delta, \emptyset)$, $\mathbf{D}[B] \neq (\Delta, \emptyset)$ and $(\mathbf{D}[A] \lceil \hat{\mathscr{A}}(B)) \supseteq \mathbf{D}[B]$, or

(2) $\mathbf{D}[A] = (\Delta, \emptyset)$.

Example. $(a \nabla b) \| ((b \nabla c); d)) \models_{\mathrm{p}} (a \| c)$.

This is a definition of partial process properties on the semantical level. However, using an algebraic process specification, we can propose some syntactical rules for the deduction of the partial properties of a process defined by an AFP_1 formula.

In the following inference rules for the deduction of partial properties we suppose that the condition $\mathscr{A}(A) \cap \mathscr{A}(B) = \emptyset$ is valid for the formulae A and B:

(I) $A \otimes B \vdash_{\mathrm{p}} A$ and $A \otimes B \vdash_{\mathrm{p}} B$, where $\otimes \in \{\|, ;, \nabla\}$.

(II) $A \otimes B \vdash_{\mathrm{p}} C \otimes D$, where $A \vdash_{\mathrm{p}} C$, $B \vdash_{\mathrm{p}} D$, $\otimes \in \{\|, ;, \nabla\}$.

Inference rules (I), (II) in the case when the formula on their left-hand side has the form of a normal $\|$-conjunct are valid without additional restrictions.

(III) Let A be in the disjunctive normal form.

Then $A = (A_1 \vee A_2 \vee \cdots \vee A_n) \vdash_{\mathrm{p}} (B_1 \vee B_2 \vee \cdots \vee B_n)$, where $\forall i$, $1 \leq i \leq k \leq n$: $A_i \vdash_{\mathrm{p}} B_i$.

A partial property B is deduced from A iff there exists a finite sequence:

$$A \vdash_{\mathrm{p}} A_1 \vdash_{\mathrm{p}} A_2 \vdash_{\mathrm{p}} \cdots \vdash_{\mathrm{p}} B$$

in which for any i, $1 \leqslant i \leqslant n$, $A_i \vdash_{\mathrm{p}} A_{i+1}$ is an application of the inference rules (I)-(III) for partial properties or $A_i = A_{i+1}$ is an application of an equivalence transformation from the set of axioms (1.1)-(6.13).

We have the following completeness theorem.

Theorem 4. *For any process formula $A \in \mathrm{AFP}_1$ the following statement is valid*:

$$A \models_{\mathrm{p}} B \Leftrightarrow A \vdash_{\mathrm{p}} B.$$

Thus, all the valid partial properties for structural processes can be deduced by means of the axiom system and the inference rules introduced above.

Examples. (1) Let $A = (a \nabla b) \| (c \nabla d)$ be a process formula. Let us prove that $(a \| c)$ is a partial property for A. The proof consists of the following steps:

$$\frac{(a \nabla b) \vdash_{\mathrm{p}} a \qquad (c \nabla d) \vdash_{\mathrm{p}} c}{(a \nabla b) \| (c \nabla d) \vdash_{\mathrm{p}} (a \| c)}\, p$$

(2) Let $A = (a \nabla b) \| ((b \nabla c); d)$ be a process formula. Let us prove that $(a \| d)$ is a partial property for A. Using the axiom system, we reduce the formula A to its canonical form and then apply the inference rules:

$$\cfrac{\cfrac{\cfrac{(a \| (c; d) \| \bar{b}) \vee ((b; d) \| \bar{a} \| \bar{c})}{(a \| (c; d) \| \bar{b})}\, p}{(a \| (c; d))}\, p}{(a \| d)}\, p$$

A property B is *total* for a process A (denotation: $A \models_{\mathrm{t}} B$) iff

(1) $\mathbf{D}[A] \neq (\Delta, \emptyset)$, $\mathbf{D}[B] \neq (\Delta, \emptyset)$ and $(\mathbf{D}[A] \lceil \hat{\mathscr{A}}(B)) = \mathbf{D}[B]$, or

(2) $\mathbf{D}[A] = [\Delta, \emptyset)$.

In the following inference rules for the deduction of total properties of processes we suppose that the condition $\mathscr{A}(A) \cap \mathscr{A}(B) = \emptyset$ is valid for the formulae A and B:

(I) $A \otimes B \vdash_{\mathrm{t}} A$ and $A \otimes B \vdash_{\mathrm{t}} B$, where $\otimes \in \{\|, ;, \nabla\}$.

(II) $A \otimes B \vdash_{\mathrm{t}} C \otimes D$, where $A \vdash_{\mathrm{t}} C$ and $B \vdash_{\mathrm{t}} D$ and $\otimes \in \{\|, ;\}$.

Inference rules (I) and (II) in the case where the formula on the their left-hand side have the form of normal $\|$-conjunct are valid without additional restrictions.

(III) Let A be in a canonical form: $A = (A_1 \vee A_2 \vee \cdots \vee A_n)$.

If $\forall i$: $1 \leqslant i \leqslant n$: $A_i \vdash_{\mathrm{t}} B_i$ then $A \vdash_{\mathrm{t}} (B_1 \vee B_2 \vee \cdots \vee B_n)$.

A total property B is deduced from A iff there exists a finite sequence

$$A \vdash_{\mathrm{t}} A_1 \vdash_{\mathrm{t}} A_2 \vdash_{\mathrm{t}} \cdots \vdash_{\mathrm{t}} B$$

in which $A_i \vdash_{\mathrm{t}} A_{i+1}$ $(1 \leqslant i \leqslant n)$ is an application of inference rules (I)-(III) for total properties for $A_i = A_{i+1}$ is an application of an equivalence transformation from the set of axioms (1.1)-(6.13).

Theorem 5. *For any process formula* $A \in \mathrm{AFP}_1$ *the following statement is valid*:

$$A \models_t B \Leftrightarrow A \vdash_t B.$$

Thus, all the valid total properties for structural processes can be deduced by means of the axiom system and the inference rules introduced above.

Examples. (1) Let $A = (a \nabla b) \| ((b \nabla c); d)$ be a process formula. Let us prove that d is a total property (i.e. that the action d may be performed during all the possible behaviours of process A). As has been noted earlier, the formula A is reduced to the following canonical form

$$A = (a \| (c; d) \| \bar{b}) \vee ((b; d) \| \bar{a} \| \bar{c}) = B \vee C.$$

Using inference rule (III) (for deduction of total properties), we prove that the following statements are valid:

$$B = (a \| (c; d) \| \bar{b}) \vdash_t (c; d) \vdash_t d,$$

$$C = ((b; d) \| \bar{a} \| \bar{c}) \vdash_t (b; d) \vdash_t d.$$

Thus, $A = B \vee C \vdash_t d$.

(2) Let $A = (a \nabla b) \| (a; (c \nabla d))$ be a process formula. Let us prove that $B = \bar{b}$ is a total property (i.e. that the action b can never occur during all the possible behaviours of the process A). Using the axiom system, we reduce the formula A to its canonical form: $A = ((a; c) \| \bar{b})$. Thus, $A = ((a; c) \| \bar{b}) \vdash_t B = \bar{b}$.

7. Abstraction

The concept of abstraction is important in the design and specification of hierarchical (or modularized) systems with communicating processes. An abstraction mechanism allows us to simplify the construction and verification of large hierarchical systems represented as systems built of smaller subsystems.

The abstraction operator λ_I which "hides" process actions from a subset $I \in \mathscr{A}$ may be introduced explicitly. Denotational semantics of λ_I may be defined as follows:

$$\mathbf{D}[\lambda_I(P)] = \lambda_I(\mathbf{D}[P])$$

where the abstraction operator λ_I is introduced for posets in the following way:

$$\lambda_I(V, <) = \begin{cases} (\delta, \emptyset) & \text{if } V \subseteq \Delta_I, \\ (V \backslash J, < \cap ((V \backslash J) \times (V \backslash J))), \end{cases}$$

where $J = I \cup \bar{I}$; i.e. if all concealed actions are deadlocked then the resulting poset(process) will be a special deadlocked process, otherwise we take a projection of the poset $(V, <)$ on the set $V \backslash J$.

Let us introduce, in addition, an empty process ε and a deadlock process δ:

$$\mathbf{D}[\varepsilon] = (\emptyset, \emptyset), \qquad \mathbf{D}[\delta] = (\{\delta\}, \emptyset).$$

If we add to the syntax and semantics of AFP_1 the empty process ε, the deadlocked process δ and the abstraction operator λ_I then the algebra AFP_1^λ arises.

In order to get the complete axiom system for AFP_1^λ corresponding to the equivalence relation $\approx_e$ (see Section 4), we will add to the axiom system introduced in Section 4 the following set of equivalences characterizing the properties of λ_I.

Axioms for the abstraction operator:

(7.1) $\lambda_I(A \vee B) = \lambda_I(A) \vee \lambda_I(B)$,
(7.2) $\lambda_I(A \| B) = \lambda_I(A) \| \lambda_I(B)$,
where $(A \| B)$ is a normal $\|$-conjunct,
(7.3) $\lambda_I(a;b) = \lambda_I(a);\lambda_I(b)$,
(7.4) $\lambda_I(a) = a$, if $a \notin I$,
(7.5) $\lambda_I(\bar{a}) = \bar{a}$, if $a \notin I$,
(7.6) $\lambda_I(\delta_a) = \delta_a$, if $a \notin I$,
(7.7) $\lambda_I(a) = \varepsilon$, if $a \in I$,
(7.8) $\lambda_I(\bar{a}) = \varepsilon$, if $a \in I$,
(7.9) $\lambda_I(\delta_a) = \varepsilon$, if $a \in I$,
(7.10) $a;\varepsilon = a$,
(7.11) $\varepsilon;a = a$,
(7.12) $\delta;a = \delta_a$,
(7.13) $a;\delta = \delta_a$,
(7.14) $\delta \| a = \delta_a$,
(7.15) $\delta \| a = \delta_a$,
(7.16) $\delta \| \delta_a = \delta_a$,
(7.17) $\delta;\delta = \delta$.

The following two theorems are valid for AFP_1^λ.

Theorem 6. *Every formula $A \in AFP_1^\lambda$ can be proved to be equal to a unique (up to isomorphism) canonical form (with the help of the axiom system with λ_I).*

Theorem 7. *For any process formulae A and B of AFP_1^λ, the following statement is valid:*

$$A \approx_e B \Leftrightarrow A =^\lambda B.$$

Thus, any valid equation between the AFP_1^λ processes may be proved with the axiom set with the abstraction operator.

8. Concluding remarks

In the previous sections, we have proposed and investigated the algebra of finite processes AFP_1 intended both for the description of nondeterministic concurrent processes and for the derivation of their behavioural properties.

The semantics of a process described by a formula of AFP_1 was defined as a set of partial orders. Each partially ordered set represents a "pure" concurrent subprocess (one of the possible process behaviours): each action either occurs in it exactly once or does not occur (because one of its alternative actions occurs). If some semantical contradiction is revealed in the AFP_1 formula (more precisely, it is discovered during some possible process behaviour), then we announce this process behaviour to be contradictory, denote it by δ and eliminate it from further consideration. Thus, a process specified by an AFP_1 formula is either δ (a deadlocked process) or it consits of completely "successful" executions.

However, another algebra AFP_2 which is different from AFP_1 by the semantics of the basic operations with respect to contradictory (deadlocked) behaviours may be considered [3].

As has been mentioned, if some semantical contradiction is revealed in the AFP_1 formula A during some process functioning, then this concrete process behaviour, as a whole, is announced to be contradictory and impossible, and it is eliminated from the semantics of A. If we consider the same process specification A as an AFP_2 formula then we will also take into account "contradictory" process behaviour, distinguishing the greatest of "non-contradictory" prefixes as possible process behaviours.

The algebras AFP_1 and AFP_2 proposed for specifying concurrent nondeterministic processes can be used as calculi for the subclasses of finite Petri nets. The basic operations of these algebras are "$\|$" (concurrency), ";" (precedence), "∇" (alternative) over the set of atomic events. The semantics of these operations could be defined in completely different ways. If we consider these operations as operations for building structured Petri nets [8], then the so-called descriptive algebra AFP_0 [3] emerges. Thus, the structures of finite Petri nets can be specified using the descriptive algebra AFP_0. However, if we consider the same formula (specifying the structure of Petri net) as a formula supplied by semantics of the algebra AFP_1 or AFP_2 then the same formula specifies the behaviour of the net.

Thus, we can use the same formula both for describing structures of Petri nets and for specifying and verifying their behaviour. So, the "structural", descriptive algebra AFP_0 can be provided with the analytical abilities of AFP_1 (AFP_2), and the analytical algebra AFP_1 (AFP_2) can be supplied with descriptive abilities of AFP_0 to specify the process structure as well as its behaviour.

References

[1] S.D. Brookes, C.A.R. Hoare and A.D. Roscoe, A theory of communicating sequential processes, *J. Assoc. Comput. Mach.* **31**(3) (1984) 560-599.

[2] L.A. Cherkasova, On models and algebras for concurrent processes, in: *Mathematical Foundations of Computer Science 1988*, Lecture Notes in Computer Science **324** (Springer, Berlin, 1988) 27-43.

[3] L.A. Cherkasova, Posets with non-actions: a model for concurrent, nondeterministic processes, Arbeitspapiere der GMD No. 403, Bonn, 1989.

[4] L.A. Cherkasova and V.E. Kotov, Structured nets, in: *Mathematical Foundations of Computer Science 1981*, Lecture Notes in Computer Science **118** (Springer, Berlin, 1981) 242-251.

[5] L.A. Cherkasova and V.E. Kotov, Descriptive and analytical process algebras, in: *Advances in Petri Nets 1989*, Lecture Notes in Computer Science **424** (Springer, Berlin, 1989) 77-104.

[6] H. Genrich, K. Lautenbach and P.S. Thiagarajan, Elements of general net theory, in: *Net Theory and Applications*, Lecture Notes in Computer Science **84** (Springer, Berlin, 1980) 21-164.

[7] U. Goltz and W. Reisig, Processes of place-transition nets, in: *Automata, Languages and Programming*, Lecture Notes in Computer Science **154** (Springer, Berlin, 1983) 164-177.

[8] V.E. Kotov, An algebra for parallelism based on Petri nets, in: *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science **64** (Springer, Berlin, 1978) 39-55.

[9] V.E. Kotov, On the basic parallel language, in: S.H. Lavington, ed., *Proc. IFIP Congress '80*, (North-Holland, Amsterdam, 1981) 229-240.

[10] V.E. Kotov and L.A. Cherkasova, On structural properties of generalized processes, in: *Advances in Petri Nets*, Lecture Notes in Computer Science **188** (Springer, Berlin, 1984) 288–306.
[11] V.E. Kotov and A.S. Narin'ani, Asynchronous processes over shared memory, *Kybernetika* **3** (1966) 64–71 (in Russian).
[12] R. Milner, *Calculus of Communicating Systems*, Lecture Notes in Computer Science **92** (Springer, Berlin, 1980).
[13] M. Nielsen, G. Plotkin and G. Winskel, Petri nets, event structures and domains, *Theoretical Computer Science* **13** (Springer, Berlin, 1981) 85–108.
[14] C.A. Petri, Non-sequential processes, GMD-ISF, Report 77-05, Bonn, 1977.
[15] C.A. Petri, Introduction to general net theory, in: *Net Theory and Applications*, Lecture Notes in Computer Science **84** (Springer, Berlin, 1980) 1–20.
[16] V.R. Pratt, Modelling concurrency with partial orders, *Internat. J. Parallel Programming* **15**(1) (1987) 33–71.
[17] W. Reisig, *Petri Nets: An Introduction* (Springer, Berlin, 1985).